Grado en Ingeniería Informática
Grado en Matemáticas e Informática

Asignatura: PROGRAMACIÓN II

# Colecciones no Acotadas

Ángel Lucas González Martínez
Jaime Ramírez Rodríguez

DLSIIS - E.T.S. de Ingenieros Informáticos
Universidad Politécnica de Madrid

Noviembre 2014

# Colecciones no Acotadas

Cadenas enlazadas

# Cadenas enlazadas

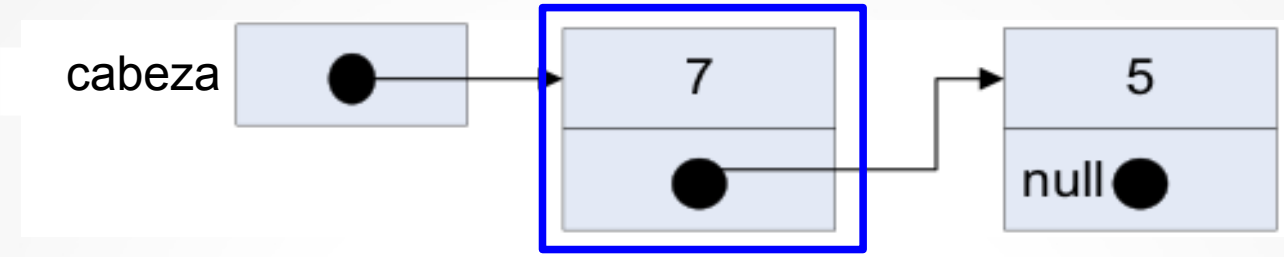
- Se utilizan para almacenar secuencias de datos cuyo tamaño máximo no se conoce al iniciarse el programa.
- Problema: ordenar una secuencia de enteros dada por el teclado.
  - ¿Dónde guardamos esta secuencia?

```
int secuencia[] = new int[TAMMAX]; // ¿TAMMAX?
```

  - No sabemos qué valor usar para TAMMAX: si usamos un valor muy grande, desperdiciaremos el espacio, y si usamos uno pequeño, nos podemos quedar cortos.
- Solución:  estructura de datos de tamaño variable.
  - El único límite vendrá dado por la memoria del ordenador
  - La estructura tendrá el tamaño necesario en cada momento, ni más ni menos.

# Cadenas enlazadas

- Ejemplo: cadena enlazada para la secuencia <7, 5>:




- La cadena está formada por nodos.
- Cada nodo contiene un dato, y una referencia al siguiente nodo (si no existe, la referencia vale **null**).
- Para acceder a los nodos de la cadena, necesitamos un puntero al primer nodo (o *cabeza*)
  - Cada vez que queramos realizar una operación sobre la cadena debemos proporcionar este puntero.
- La cadena vacía se representa como una referencia con valor **null**.

# Implementación en Java

```
public class NodoEntero {
  private int dato;
  private NodoEntero siguiente;
  public Nodo(int dato) {
    this.dato=dato;
    this.siguiente=null;
  }
  public NodoEntero(int dato, NodoEntero siguiente) {
    this.dato=dato;
    this.siguiente=siguiente;
  }
  public NodoEntero getSiguiente() {
    return this.siguiente;
  }
  public int getDato() {
    return this.dato;
  }
  public void setSiguiente (NodoEntero siguiente) {
    this.siguiente=siguiente;
  } }
```

# Implementación en Java

```
public class CadenaEnteros {
  private NodoEntero cabeza;
  public CadenaEnteros() { // constructor: crea cadena vacía
    this.cabeza = null;
  }
  public void setCabeza(NodoEntero cabeza) {
    this.cabeza = cabeza;
  }
  public NodoEntero getCabeza() {
    return this.cabeza;
  }
}
```

# Ejemplo de manejo (I)

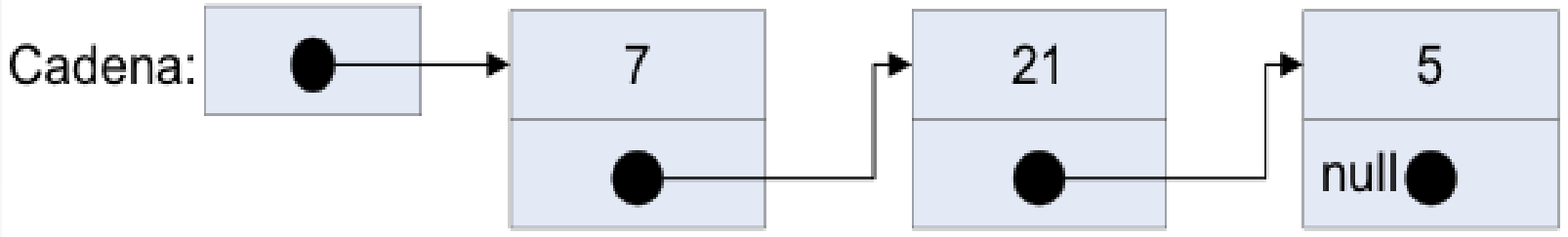
- El código para crear la cadena enlazada para la secuencia <7, 5> sería:

```
public class PruebaCadena {
  public static void main(String[] args) {
    CadenaEnteros cadena = new CadenaEntero();

    cadena.setCabeza(new NodoEntero(7, null));
    NodoEntero cabeza = cadena.getCabeza();
    cabeza.setSiguiente(new NodoEntero(5, null));
    // equivale a esto:
    // cadena.setCabeza(
         new NodoEntero(7, new NodoEntero(5, null))
       );
    }
  }
}
```

# Ejemplo de manejo (II)

- Supongamos que queremos introducir un elemento en la cadena anterior <7, 5> y obtener <7, 21, 5>:



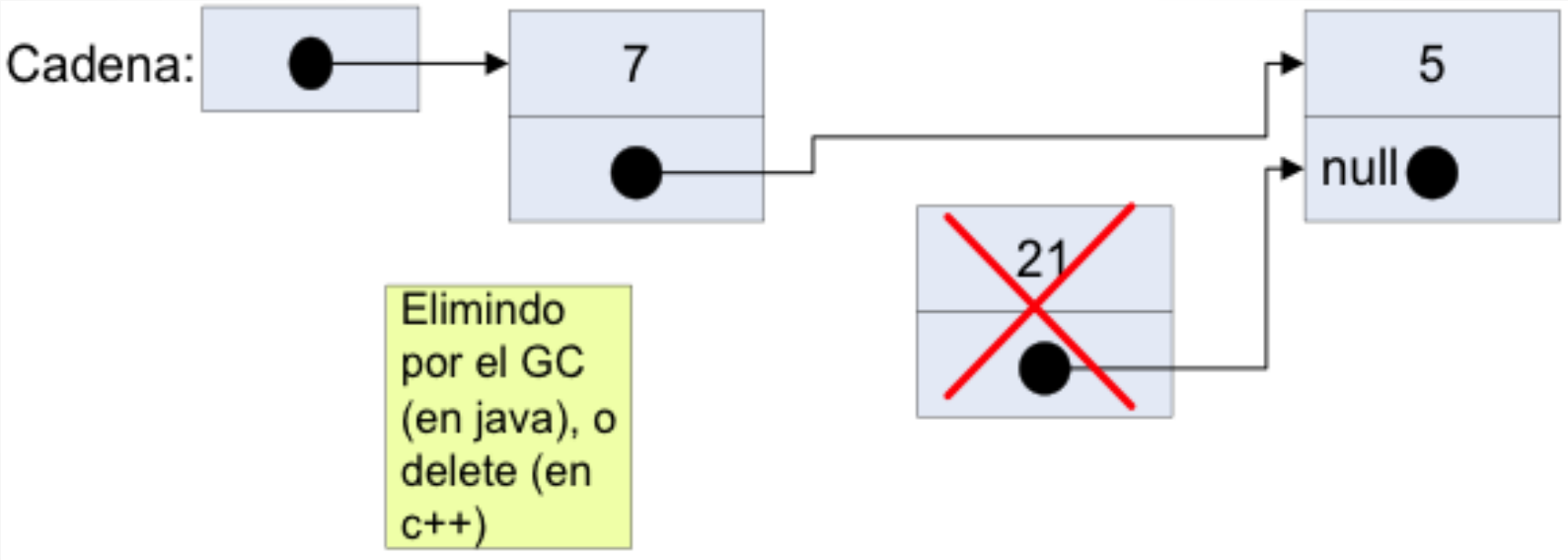

- El código podría ser el siguiente:

```
cabeza = cadena.getCabeza();
cabeza.setSiguiente(
   new NodoEntero(21, cabeza.getSiguiente())
);
```

# Ejemplo de manejo (III)

- Y para eliminarlo y obtener de nuevo < 7, 5 >:

```
cabeza = cadena.getCabeza();
cabeza.setSiguiente(cabeza.getSiguiente().getSiguiente());
```



# Colecciones no Acotadas

Operaciones sobre Cadenas Enlazadas

# Operaciones que no recorren la cadena

- Operaciones observadoras: esVacia(), getPrimero()
- Inserción al principio de la cadena
- Borrado del primer elemento
- Vaciar la cadena

# Operaciones que recorren la cadena

- Recorrido completo
  - toString()
  - Obtener la longitud de la cadena
  - Inserción al final de la cadena
  - Borrado del último elemento
- Recorrido parcial
  - Buscar un elemento dado en la cadena
  - Inserción en la n-ésima posición de la cadena
  - Borrado del n-ésimo elemento de la cadena

# Otras clases de cadenas enlazadas

- Las cadenas enlazadas se emplean para implementar secuencias.
- Las cadenas enlazadas vistas hasta ahora presentan algunas carencias con respecto a la eficiencia de algunas de sus operaciones (por ejemplo, longitud, inserción por el final, etc.).
- Existen distintas variantes de las cadenas simplemente enlazadas para eliminar esas desventajas:
  - Mantener la **longitud como dato precalculado** (cadenas enlazadas con longitud).
  - Mantener un **puntero al último nodo** de la cadena (cadenas enlazadas con puntero al último y cadenas enlazadas circulares).
- Otra variante: cadenas doblemente enlazadas
  - Permiten recorrer la cadena en los dos sentidos

# Cadenas con longitud precalculada

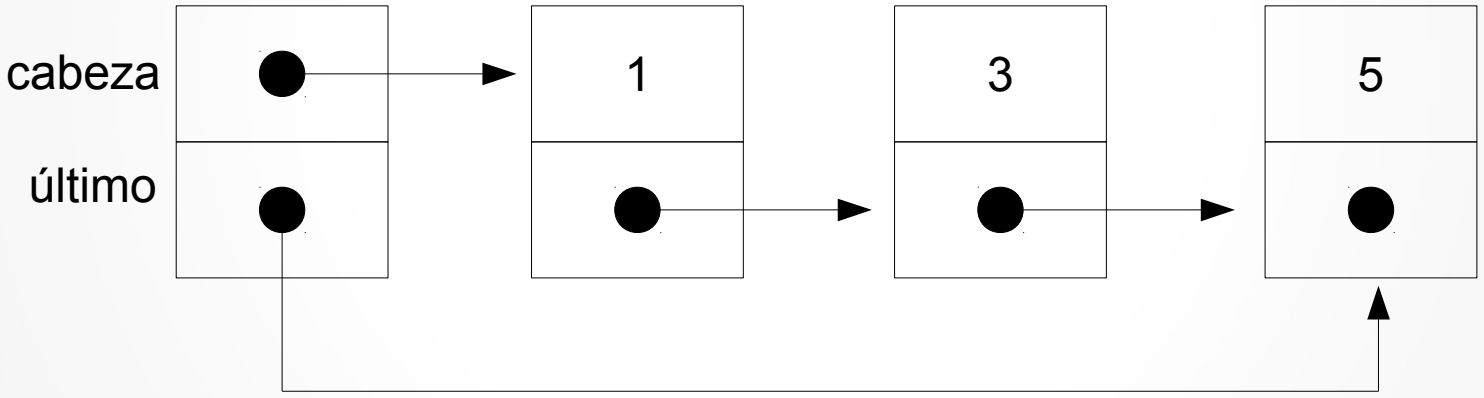
- Se utilizan para evitar tener que recorrer la cadena entera cada vez que se necesite calcular la longitud de la cadena.
- Se añade un atributo **contadorElems** a la clase CadenaEnteros
- Se inicializa a cero cuando se crea la cadena vacía o se vacía la cadena
- Se incrementa **contadorElems**, cada vez que se añada un nuevo elemento a la cadena.
- Se decremente **contadorElems**, cada vez que se elimine un elemento de la cadena.
- El constructor copia asigna el número de elementos de la cadena dada a la cadena **this**.

# Cadenas con puntero al último

- Hay algunas operaciones de la cadena enlazada que requieren disponer de una referencia al último nodo de la cadena: insertar al final, borrar el último, etc.
- Para poder obtener una referencia al último nodo, necesitamos recorrer la cadena entera.
  - Necesitamos un bucle para recorrer la cadena
- Es más rápido si disponemos de un acceso directo (referencia) al último nodo.
  - Nos ahorramos recorrer la cadena entera

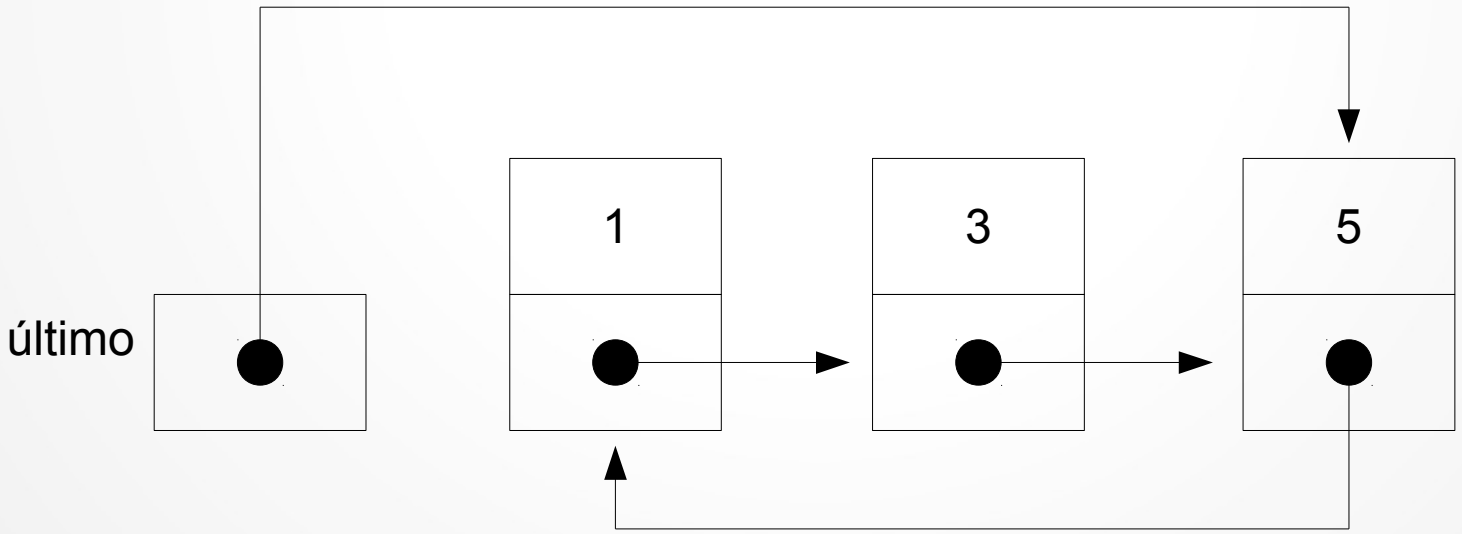
# Cadenas con puntero al último

- SOLUCIÓN: añadimos un nuevo atributo **ultimo** a la clase CadenaEnteros
  - La clase CadenaEnteros tiene dos atributos: **cabeza** y **ultimo**.



# Cadenas circulares

- Pero si disponemos de una referencia al último nodo, nos podemos ahorrar la referencia al primero.
- Definimos una cadena circular en la que el último nodo apunta al primero en vez de apuntar a **null**.
  - Solo necesitamos el atributo **ultimo**
- Si la cadena tiene un sólo nodo, este nodo se apuntará a sí mismo.

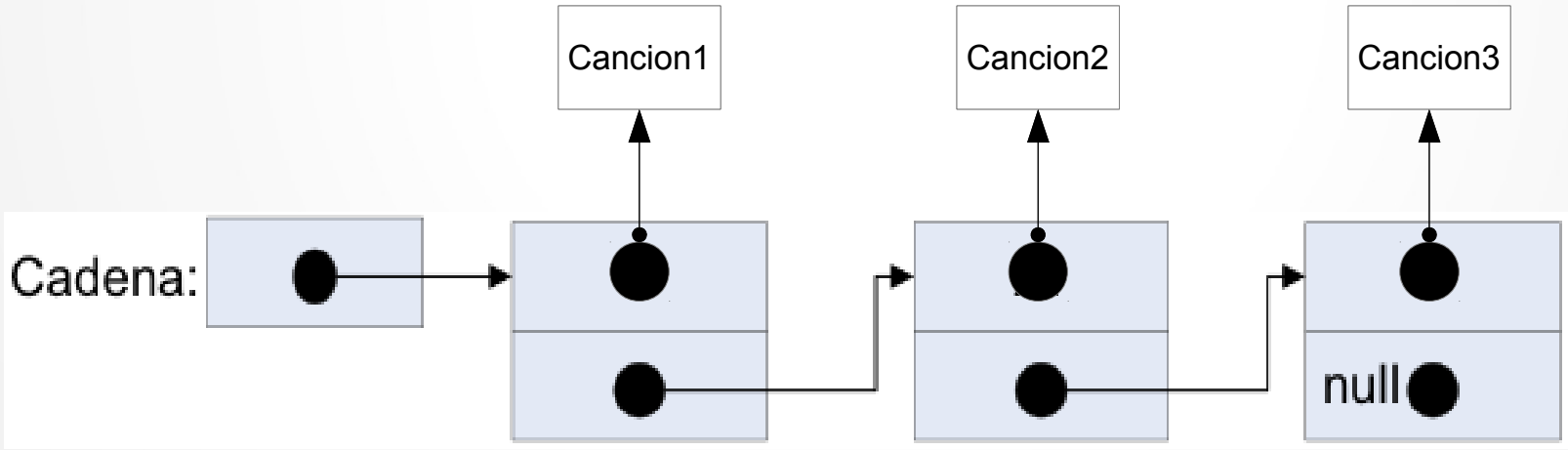


# Colecciones no Acotadas

Cadenas Enlazadas de Objetos

# Cadenas de objetos

- Hasta ahora la cadena con la que hemos trabajado contenía como dato un tipo básico (int).
- Es relativamente fácil de modificar la clase CadenaEntero para que nos sirva para guardar datos de tipo objeto (fechas, personas, coches, etc.).
- Ejemplo: Cadena enlazada de canciones



# Cadenas de objetos

- Si en la cadena el dato es de tipo objeto, cada nodo contendrá una referencia a un objeto.
  - Esto va a afectar a la implementación de varias operaciones:
    - Todas las que comparen dos datos para comprobar si son iguales: == → esIgual()
    - constructor copia

# Métodos que utilizan la igualdad en una cadena de objetos

- esIgual(), buscarElemento(dato)
- La única diferencia con los métodos de la clase CadenaEnteros es:
  - Se utiliza el método esIgual() para comparar dos datos en lugar del operador de igualdad (==)
  - Debe existir el método esIgual() en la clase que define el dato.

# Constructor Copia para una cadena de objetos

- Existen dos maneras de crear una copia de una cadena de objetos
  - *Shallow copy*: la cadena copia comparte los objetos dato con la original.
    - Se utiliza el operador de asignación al inicializar los objetos datos de la cadena copia
  - *Deep copy*: la cadena copia tiene sus propios objetos dato.
    - Se utiliza el constructor copia al inicializar los objetos datos de la cadena copia

# Constructor Copia para una cadena de objetos

- *Shallow Copy* Vs *Deep Copy*